Nombre del trabajo: Foro 1

Materia: PROGRAMACIÓN ORIENTADA A OBJETOS POO941

Alumnos:
Henry Alejandro Martínez Guerra

Steven Benjamin Diaz Flores

Christian Alexander Hernandez Funes

José Alexander Sermeño Zetino

Docente: Ing.Rafael Alexander Torres Rodríguez

Fecha de entrega:

20 de agosto de 2020

Introducción.

JAVA es un lenguaje de programación orientado a objetos, en ese contexto Collections (Colecciones) representan un marco de trabajo estructurado y eficiente que existe con el propósito de facilitar el trabajo de programación, el uso de las interfaces de las colecciones reduce considerablemente el esfuerzo de programación, ya que contienen muchas estructuras y algoritmos útiles y esto permite que el esfuerzo sea enfocado en resolver las partes importantes de un programa en lugar de perder el tiempo en imaginar la manera de adaptar herramientas menos sofisticados, bajo esa misma lógica reduce el tiempo de programación y aumenta la calidad de los programas generados, el tiempo se utiliza mejor ya que no debemos desperdiciar nuestro trabajo en crear estructuras de datos complejas por que ya existen dentro de JAVA.

Entre estas características nos resulta clara la versatilidad de JAVA no solo como lenguaje de programación si no como herramienta para incrementar la productividad y facilitar el trabajo entre programadores, por lo que se revisarán las características generales de colecciones fundamentales como Map y List, la forma general en el que deben declararse cada una de sus implementaciones y como agregar o eliminar información de las mismas. A partir de estos conocimientos se desarrollará un pequeño programa que realiza una tarea sencilla utilizando dichas estructuras.

Interfaz Collection

Una colección (“Collection”) en JAVA representa un grupo de objetos, conocidos como sus *elementos*. Algunas colecciones permiten elementos duplicados y otras no, algunos están ordenados y otros desordenados. El JDK no proporciona ninguna implementación directa de esta interfaz: proporciona implementaciones de subinterfaces más específicas como Set, List, Queue, Deque) y clases (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet.

Todas las clases de implementación de propósito general de Collection deben proporcionar dos constructores "estándar": un constructor Void (sin argumentos), que crea una colección vacía, y un constructor con un solo argumento de tipo Collection, que crea una nueva colección con los mismos elementos que su argumento. El último constructor permite al usuario copiar cualquier colección, produciendo una colección equivalente del tipo de implementación deseado.

Las operaciones básicas de una collection son:

- add, añade un elemento.
- iterator, Obtiene un “iterador” que permite recorrer la colección visitando cada elemento una vez.
- Size, Obtiene la cantidad de elementos que esta colección almacena.
- Contains, Pregunta si el elemento *t* ya está dentro de la colección.

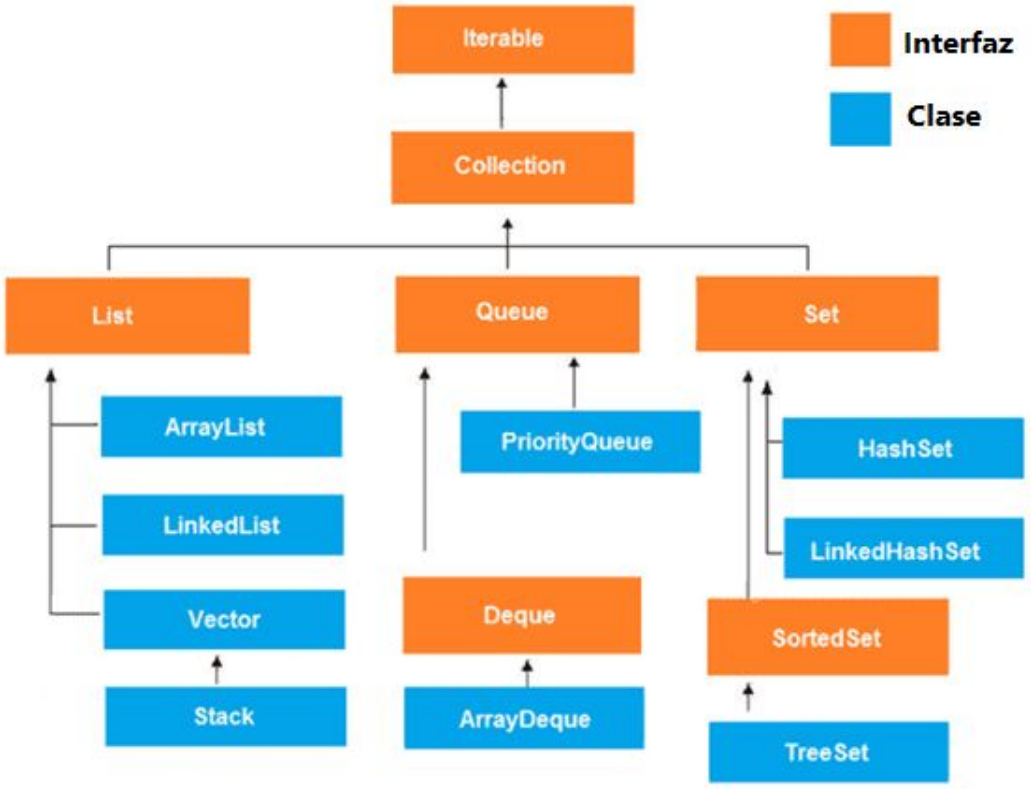

Figura 1. Diagrama general de la estructura de JAVA Collections.

Cómo declarar una Collection

Java Collection Framework es la librería de clases contenedoras donde está el paquete estándar java.util. Estas clases sirven para almacenar colecciones de objetos como List, Maps, etc. Todas las colecciones genéricas proporcionan funcionalidades básicas como los métodos add y remove. y se declaran siguiendo la siguiente estructura:

```
public interface Collection<E>
```

donde "E" es el tipo de objeto contenido en la colección Java.

Agregar o eliminar valores dentro de una Colección

Como cualquier interfaz en Java para poder agregar, eliminar datos, Collection nos obliga a implementar varios métodos de los cuales destacan los siguientes:

- Boolean add (E e)
- Boolean remove (Object o)

Una capacidad importante de un objeto Collection es el uso de iteradores. Un iterador es un objeto "paseador" que nos permite obtener todos los objetos al ir invocando progresivamente el método next. Si la colección es modificable, podemos remover un objeto durante el recorrido mediante el método remove. El siguiente ejemplo recorre una colección Integer borrando todos los ceros:

```
void borrarCeros(Collection<Integer> ceros){
Iterator<Integer> it = ceros.iterator();
        while(it.hasNext())
        {
        int i = it.next();
        if(i == 0)
                it.remove();
        }
}
```

A partir de Java 6 hay una manera simplificada de recorrer una collection (que sirve si no necesitamos borrar elementos). Se hace mediante un nuevo uso del keyword for:

```
void mostrar(Collection<?> col)

{

        for(Object o : col)

        System.out.println(o);

}
```

La interfaz Lista

Una lista es una colección ordenada (a veces llamada secuencia). Las listas pueden contener elementos duplicados. Además de las operaciones heredadas de Collection, la interfaz List incluye operaciones para lo siguiente:

- Acceso posicional: manipula elementos en función de su posición numérica en la lista. Esto incluye métodos como get, set, add, addAll, and remove.
- Buscar: busca un objeto específico en la lista y devuelve su posición numérica. Los métodos de búsqueda incluyen indexOf y lastIndexOf.
- Iteración: amplía la semántica del iterador para aprovechar la naturaleza secuencial de la lista. Los métodos listIterator proporcionan este comportamiento.
- Vista de rango: el método de sublis realiza operaciones de rango arbitrarias en la lista.

La plataforma Java contiene dos implementaciones List de propósito general. ArrayList, que suele ser la implementación de mejor rendimiento, y LinkedList, que ofrece un mejor rendimiento en determinadas circunstancias.

Declarar una lista.

Para declarar una lista utilizando la colección “List” se debe seguir el procedimiento siguiente,

- Es necesario importar el paquete “java.util” que contiene el framework de las clases colección entre otros objetos.
- Luego dentro de la clase de código que se desea desarrollar se inicializa la colección List, es necesario especificar el tipo de dato que almacenará la lista, esto se hace utilizando el operador diamante <> y declarar el nombre de la lista.
- Luego se selecciona la implementación de la colección seleccionada y de nuevo con el operador de diamante se selecciona el tipo de dato a ingresar o manipular, finalmente entre paréntesis se ingresa el nombre de la variable o arreglo donde están alojados los valores o elementos a manipular.

La declaración genérica explicada anteriormente sería la siguiente, donde (c) podría ser el nombre de un arreglo:

```
List<String> list = new ArrayList<String>(c);
```

Agregar valores en una lista.

Para asignar valores dentro de una lista utilizamos los operadores posicionales disponibles dentro de la colección “List”, estos métodos son get, set, add, addAll.

- set, Reemplaza el elemento en la posición especificada en la lista con el elemento especificado.
- add, inserta un nuevo elemento en la lista inmediatamente antes de la posición actual del cursor.
- addAll, inserta todos los elementos de la colección especificada comenzando en la posición especificada.

Ejemplos de la implementación de dichos métodos son los siguientes:

```
List<String> ejemploLista = new ArrayList<String>();

    ejemploLista.get(0); Donde 0 es el índice de la posición que será sustituida.

List<String> ejemploLista = new ArrayList<String>();

    ejemploLista.add("Juan");

List<String> ejemploLista3 = new ArrayList<Type>(ejemploLista1);

    ejemploLista3.addAll(ejemploLista2);
```

Eliminar valores en una lista.

Para eliminar valores de una lista se utiliza el operador remove.

- remove, siempre elimina la primera aparición del elemento especificado de la lista.
- removeAll, quita de esta lista todos sus elementos que están contenidos en la colección especificada (operación opcional).

Ejemplos de la implementación de remove:

```
List<String> ejemploLista = new ArrayList<String>();

    ejemploLista.remove(0); Donde 0 es el índice de la posición que será
    sustituida.

List<Character> list = new ArrayList<Character>();

    collection.removeAll(list);
```

En el siguiente enlace se encuentra un programa que realiza un listado de estudiantes según lo solicitado por la ruta de aprendizaje de la materia:

- Ejemplo Registro UDB Virtual List

La interfaz Map

Un mapa es un objeto que asigna claves a valores. Un mapa no puede contener claves duplicadas: cada clave se puede asignar como máximo a un valor. La interfaz de Map incluye métodos para operaciones básicas (como put, get, remove, containsKey, containsValue, size, y empty), operaciones masivas (como putAll y clear) y vistas de colección (como keySet, entrySet, y values).

La plataforma Java tiene tres implementaciones de mapas de uso general:

- HashMap, Implementación basada en tablas hash de la interfaz Map. Esta clase no garantiza el orden del mapa; en particular, no garantiza que el pedido se mantenga constante en el tiempo.
- TreeMap, Esta implementación proporciona un costo de tiempo log (n) garantizado para las operaciones containsKey, get, put y remove.
- LinkedHashMap, Implementación de tabla hash y lista enlazada de la interfaz Map, con orden de iteración predecible. Esta implementación se diferencia de HashMap en que mantiene una lista doblemente vinculada que se ejecuta en todas sus entradas.

Declaración de la interfaz Map

Las implementaciones de la interfaz Map debe importarse “java.util” más el tipo de mapa que se desea, luego se define el tipo de la "clave" luego el tipo del "valor" Se asigna un nombre y se crea una instancia de Map. Un modelo genérico puede observarse a continuación.

```
public static Map<String, String> articleMapOne;

    static {

    articleMapOne = new HashMap< >();
```

Asignar valores en un Map

Para asignar valores dentro de una implementación de Map se requiere utilizar el método put y se pasan las "claves" y los "valores".

```
public static Map<String, String> articleMapOne;

    static {

    articleMapOne = new HashMap<>();

    articleMapOne.put("ar01", "Intro to Map")

    articleMapOne.put("ar02", "Some article");

}
```

Cómo se eliminan los valores en la interfaz Maps

Se utiliza el método "remove" y se ingresa la “clave” que sirve como índice para eliminar el valor relacionado con dicha clave. También se puede borrar todos los elementos del HashMap con "clear". Para recorrer un HashMap existen diferentes opciones Se puede utilizar un “for-each” convencional o funcional también se puede imprimir solo las llaves o solo los valores.

El siguiente enlace presenta un ejemplo genérico realizado implementado la colección Map:

Ejemplo Map

Conclusiones

Sin lugar a dudas Collections es una herramienta poderosa y una manera muy eficiente de programar, nos permite enfocarnos en resolver los problemas y dedicar más tiempo en buscar soluciones en lugar de gastarlo en trabajar estructuras y métodos cada vez que encontremos un problema nuevo.

Se considera que esta es la fortaleza de la programación orientada a objetos, las clases y las interfaces disponibles en JAVA nos facilitan las tareas y por lo tanto podemos buscar soluciones más creativas y funcionales.

A partir de los contenidos explorados fue posible elaborar un programa sencillo capaz de hacer un registro de estudiantes, tarea que siguiendo métodos de programación estructura hubiera requerido un esfuerzo mucho mayor.

Referencias

[1]"The List Interface (The Java™ Tutorials > Collections > Interfaces)", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html. [Accessed: 21- Aug- 2020].

[2]"Collections in Java - javatpoint", *www.javatpoint.com*, 2020. [Online]. Available: https://www.javatpoint.com/collections-in-java. [Accessed: 21- Aug- 2020].

[3]"Collections en Java", *Reloco.com.ar*, 2020. [Online]. Available: http://www.reloco.com.ar/prog/java/collections.html. [Accessed: 21- Aug- 2020].

[4]J. Acedo, "Colecciones y tipos genéricos en Java – Apuntes de Programación", *Programacion.jias.es*, 2020. [Online]. Available: http://programacion.jias.es/2011/10/colecciones-genericos-en-java/. [Accessed: 21- Aug- 2020].

[5]M. Cuenca, "aprenderaprogramar.com", *Aprenderaprogramar.com*, 2020. [Online]. Available: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=596:interface-collection-api-java-add-remove-size-ejemplo-arraylist-diferencia-con-list-streams-cu00917c&catid=58Itemid=180. [Accessed: 21- Aug- 2020].

[6]P. Corcuera, *Collections*, 1st ed. Cantabria: Universidad de Cantabria, 2016.

[7]"The List Interface (The Java™ Tutorials > Collections > Interfaces)", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html. [Accessed: 21- Aug- 2020].

[8]"List (Java Platform SE 8)", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/List.html. [Accessed: 21- Aug- 2020].

[9]"El uso de Listas en Java | Panama Hitek", *Panama Hitek*, 2020. [Online]. Available: http://panamahitek.com/el-uso-de-listas-en-java/. [Accessed: 21- Aug- 2020].

[10]"Java Collection removeAll() Method with Examples - Javatpoint", *www.javatpoint.com*, 2020. [Online]. Available: https://www.javatpoint.com/java-collection-removeall-method. [Accessed: 21- Aug- 2020].

[11]"List (Java Platform SE 8)", Docs.oracle.com, 2020. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/List.html. [Accessed: 21- Aug- 2020].

[12]"HashMap (Java Platform SE 8)", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html. [Accessed: 21- Aug- 2020].

[13]"TreeMap (Java Platform SE 8)", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html. [Accessed: 21- Aug- 2020].

[14]"LinkedHashMap (Java Platform SE 8)", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html. [Accessed: 21- Aug- 2020].